SE499 Report — Path Following Controllers

Rollen S. D'Souza
Software Engineering Undergraduate
Department of Electrical & Computer Engineering
rs2dsouz@edu.uwaterloo.ca

Acknowledgments

The author thanks Professor Christopher Nielsen for his guidance in developing the required intuition and mathematical tools for path following control design. Plots and simulations were partly written in Mathworks MATLAB under a student licence. Other simulations were written using C++ linked with the Boost library[2] and Catch test framework[8].

Rollen S. D'Souza
Software Engineering Undergraduate
University of Waterloo
rs2dsouz@edu.uwaterloo.ca

Contents

Nomenclature

$B_\epsilon(\mathbf{x})$	Open ball of radius ϵ centred about $\mathbf{x}$ in the appropriate vector space.
$\mathbf{R}_\theta$	Counter-clockwise rotation matrix in $\mathbb{R}^2$ of angle θ.
$\mathcal{T}$	Mobile robot task space.
$\mathcal{T}_{(w,h)}$	Rectangular mobile robot task space with width w and height h.
$\mathbf{M}$	A matrix. Can be assumed real unless stated otherwise.
$\mathbf{M}_{(i,j)}$	The value at the i-th row and j-th column of matrix $\mathbf{M}$. The variables may have a : substituted to indicate selection of all values in that dimension, similar to that found in the Matlab grammar.
$\mathbf{x}$	A vector in $\mathbb{R}$.
$\mathbf{x}_i$	The i-th component of vector $\mathbf{x}$

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A common control objective for mobile robots involves tracking a trajectory in the space the robot operates in. This space is defined as the task space of the robot, denoted as $\mathcal{T}$. Often the structure of this space is unknown — along with the goal — and the robot must first explore the world using an exploration and mapping algorithm before deciding on a goal and path. This paper is instead concerned with a restricted subset of this problem.

Consider a differential drive robot starting at location $(0, 0)$, placed in a world with an unknown number of obstacles, that is given the objective to reach position (x, y) in task space without colliding with obstacles. A two-pronged approach can be taken to safe-guard from collisions:

1. Plan a path that can be followed successfully by the robot and does not intersect with any obstacles for all future time.

2. Design a controller that ensures sufficiently fast convergence to the path and provides a guarantee, under reasonable assumptions, that the robot will not leave the path.

Planning a path is a field in its own right. The most practical, and dominant, philosophy to path planning involves sampling $\mathcal{T}$ and incrementally building a path to the goal region. The sampling algorithm is critical to performance and effective coverage of $\mathcal{T}$. If the sampling technique does not adequately sample the space in question the algorithm may take considerably longer to generate a path that reaches $\mathcal{G}$. Incorporated in these algorithms is the ability to test the feasibility of a path such as whether any collisions would occur with the known environment and whether any other differential constraints of the robot are broken. This addition ensures that infeasible paths are rejected early instead of wasting time developing a path that is known to be infeasible. Of course, the astute reader notes that this type of path generation often creates discrete segments that lead to the goal. There are techniques in literature to develop smooth paths incrementally in coordination with a sampling algorithm but these are considered outside the scope of this work [1]. Other advanced approaches to generate smooth paths exist that rely on construction of a vector field over $\mathcal{T}$ [6] but these are relatively complex.

Instead this report takes a simpler approach that generates a smooth function out of the discrete waypoints generated by the relatively simple Rapidly exploring Randomized Tree (RRT) algorithm. The smooth function generated is a polynomial spline, a stitched sequence of possibly distinct polynomials that preserve a set of continuity conditions at the stitch-points. After generation, the spline is tested again for feasibility before being propogated to the controller. There are apparent performance issues in this approach addressed in Section 3.

Independently the field of non-linear control theory developed techniques to follow arbitrary paths. Standard control techniques tend to yield poor results in tracking a path. A number of techniques in literature exist to drive a robot towards a path and converge upon it. However, most of these techniques do not provide invariance once on the path[3]. This lack of invariance reduces the robustness of the technique as the controller may not

stay on the path even under perfect environmental conditions! This motivated the development of a new family of path following controllers that treat the path as an invariant set that is then stabilized through linear control, described in Section 2. This family of controllers use transverse feedback linearization to reformulate the problem and assist in constructing a linear controller that compensates for the non-linearities imposed by the path.

This report discusses a few methods to planning and control for the purpose of driving a differential drive robot towards a goal.

1.2 Notation and Report Organization

This report assumes proficiency in multi-variable calculus and basic control theory. As a result, any results taken from theory of dynamical systems and other advanced undergraduate mathematical courses are stated and cited from a source used by the author. Scalars are denoted as x, vectors as $\mathbf{x}$ and matrices as $\mathbf{X}$. Time derivatives are denoted using Newton's dot notation and any other derivatives are denoted using primes if the independent variable is clear and Leibniz notation otherwise.

In practice, planning algorithms generally precede control. However this report instead defers discussion of planning until after describing control methodologies in Section 2. This content placement is intended to give the reader a better intuition for the decisions made at planning stage.

1.3 Problem Statement

Let the task space, without loss of generality, be the rectangular space $\mathcal{T}_{(w,h)} = \{(x, y) \in \mathbb{R}^2 : 0 \leq x \leq w \wedge 0 \leq y \leq h\}$. For simplicity, consider the kinematic model of a differential drive robot with the combined position and orientation state vector $\mathbf{x} = (x, y, \theta)^\intercal$. The robot has a, possibly controllable, forward velocity $v : (\mathbf{x}, t) \mapsto \mathbb{R}$ with $v > 0$ and a controllable turning rate $u : (\mathbf{x}, t) \mapsto \mathbb{R}$. The observation is simply the robot's current position. Unless stated otherwise, v is assumed constant. The kinematic dynamics follow as,

$$\begin{aligned}
\mathbf{f}(\mathbf{x}) &= \begin{bmatrix} v\ cos(\mathbf{x}_3) \\ v\ sin(\mathbf{x}_3) \\ 0 \end{bmatrix} \\
\mathbf{g} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
\mathbf{H} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \\
\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \mathbf{g}u \\
\mathbf{h}(\mathbf{x}) &= \mathbf{H}\mathbf{x}
\end{aligned} \tag{1.1}$$

The robot, at some unknown finite time, must arrive at the goal region $\mathcal{G} \subset \mathcal{T}$. Note there is no restriction in how it arrives in this region other than it does so safely. Starting at time $t = 0$ and then at a regular time interval Δ_p, the robot observes the world and decides, based on the observation, whether a new plan is required to approach the goal region. This may occur, for example, when the robot observes a new obstacle that the robot's currently planned path intersects. The simplest and most widely used planning algorithms are involve the generation of a tree that explores the current model of the world. This family of algorithms generate a sequence of waypoints. This is then used to generate a path $\sigma : \lambda \mapsto \mathcal{T}$ where λ is an arbitrary parameterization of the curve. Without loss of generality, this report assumes $\lambda \in [0, 1]$.

The path σ_1 is fed into the path following control which guides the robot over time towards, and along, the path in order to arrive in $\mathcal{G}$. Further, the control algorith must converge towards the path in a manner that is both guaranteed and as fast as possible. It must also faithfully follow the path, in order to reduce the

chances of replanning. Given that the robot may not have a perfect model of the world, i.e. is not aware of all obstacles, a replan may still occur. The planning algorithm is required to design the new σ_2 so as to preserve any properties the control algorithm requires to stay on the trajectory continuously.

Chapter 2

Control

Three control strategies are considered for path following. The simplest controller, in terms of design and implementation, is the point-chasing controller. The other two techniques rely on the approach of transverse feedback linearization which decomposes the path following problem into transverse and tangential dynamical systems.

This section assumes the path is of the form in Equation 2.1. The matrix representation of a quintic polynomial is chosen for succint representation.

$$\begin{aligned} &\mathbf{P} \in \mathbb{R}^{2\times 6} \\ &\sigma(\lambda) = \mathbf{P} \begin{bmatrix} 1 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \\ \lambda^4 \\ \lambda^5 \end{bmatrix} \end{aligned} \tag{2.1}$$

2.1 Point-Chasing (Tracking) Controller

A point-chasing (tracking) controller involves treating the parameter λ of the path σ as a function of time, t, and designing a controller that converges to this moving point. In order to apply this strategy, the control is designed around a point that leads the robot's current position. That is, consider an observation $\mathbf{y}_l(\mathbf{x}(t)) = \mathbf{H}\mathbf{x}+l\mathbf{tau}(\mathbf{x}_3)$ for some $l \in \mathbb{R}$ and $l > 0$, where $\mathbf{tau}(\mathbf{x}_3) = (cos(\mathbf{x}_3), sin(\mathbf{x}_3))^\intercal$. We let $\mathbf{y}_{ref}(t) = \sigma(\lambda(t))$. The introduction and restriction on the parameter l is clear in the derivation.

There is one modification made to the differential drive robot to allow for this controller design. The differential drive robot must permit speed control. Therefore we take, for this sub-section alone, the dynamical system to be of the form,

$$\dot{\mathbf{x}} = \mathbf{f}_t(\mathbf{x}) = \begin{bmatrix} v\ cos\ \mathbf{x}_3 \\ v\ sin\ \mathbf{x}_3 \\ u \end{bmatrix} \tag{2.2}$$

Define the position error $\mathbf{e} = \mathbf{y}_l - \mathbf{y}_{ref}$ and take the time derivative to form a new dynamical system in

terms of the path error,

$$\begin{aligned}\dot{\mathbf{e}} &= \dot{\mathbf{y}}_\mathbf{l} - \dot{\mathbf{y}}_\mathbf{ref} \\ &= \mathbf{H}(\mathbf{f}(\mathbf{x})) + l\dot{\tau} - \sigma'(\lambda(t))\dot{\lambda} \\ &= \begin{bmatrix} v\ cos\ \mathbf{x}_3 \\ v\ sin\ \mathbf{x}_3 \end{bmatrix} + \begin{bmatrix} -l\ sin\mathbf{x}_3 \\ l\ cos\ \mathbf{x}_3 \end{bmatrix} u - \sigma'(\lambda(t))\dot{\lambda} \\ &= \begin{bmatrix} -l\ sin\mathbf{x}_3 & cos\ \mathbf{x}_3 \\ l\ cos\ \mathbf{x}_3 & sin\ \mathbf{x}_3 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - \sigma'(\lambda(t))\dot{\lambda}\end{aligned}$$

At this point the reader may observe that the matrix multiplying the control signals is invertible if and only if l is non-zero. Let the dynamics of $\dot{\mathbf{e}}$ be governed by proportional feedback control. That is, $\dot{\mathbf{e}} = \mathbf{Ke}$ with $\mathbf{K} \in \mathbb{R}^{2\times 2}$ and Hurwitz. Then the control signals may be solved for,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -l\ sin\mathbf{x}_3 & cos\ \mathbf{x}_3 \\ l\ cos\ \mathbf{x}_3 & sin\ \mathbf{x}_3 \end{bmatrix}^{-1} \left[\mathbf{Ke} + \sigma'(\lambda(t))\dot{\lambda}(t)\right]$$

2.2 Transverse Feedback Linearization

A variety of techniques were designed to address the issues observed in the point chasing controller. One of these techniques is transverse feedback linearization. Transverse feedback linearization is the mathematical reformulation of system dynamics into transversal and tangential components with respect to the closest point to the path. This restatement of the problem allows for the capturing of the path dynamics — combining that with the original system dynamics — and permits cancellation of non-linearities introduced by the path as it affects the system.

The technique was generalized in [9] for aperiodic paths and then further expanded in [**Hladio13**]. The approaches defined in those works restricted their design to following simple curves, such as circles. Gill et al., in [3], applied the technique to a larger class of paths, specifically splines in C^2. Note that their work is not restricted to polynomial splines and in fact applies to any functions that are stitched at waypoints preserving C^2 continuity on the whole parameterized domain. As stated previously, it is assumed in this work that these splines are quintic polynomials in order to provide a relevant benchmark comparison.

2.2.1 General Form

Recall the kinematic model for a differential drive robot, from Equation 1.1.

As before, let the path $\sigma : \lambda \mapsto \mathbb{R}^2$ with the additional requirement that it be C^2 everywhere. We define a new state vector $\xi = (\xi_1, \xi_2)^\intercal$ where ξ_1 is left undefined for now. One can consider ξ_1 as the signed tracking error of the robot with respect to the path. The variation of this state is what differentiates the variety of techniques used. When the path is well known and simple — such as a circular arc — this state may be expressed in closed form. This is not always the case. Let us then define $\xi_2 = \dot{\xi}_1$. Chain rule gives,

$$\begin{aligned}\xi_2 &= \dot{\xi}_1 \\ &= \frac{\partial \xi_1}{\partial \mathbf{x}}\dot{\mathbf{x}} \\ &= \nabla_\mathbf{x}\xi_1 \begin{bmatrix} v\ cos(\mathbf{x}_3) \\ v\ sin(\mathbf{x}_3) \\ u \end{bmatrix}\end{aligned}$$

Under the assumption that this error is independent of robot heading we can further simplify the expression such that $\xi_2 = \nabla_\mathbf{x}\xi_1 f(\mathbf{x}) = \mathcal{L}_f\xi_1$. $\mathcal{L}_f\xi_1$ is known as the Lie derivative of ξ_1 with respect to the vector field $f(\mathbf{x})$. This then means the control signal has no impact on our state. This motivates finding the time derivative of

ξ_2. Using properties of Lie derivatives, it is not difficult to show that $\xi_2 = \mathcal{L}_{gf}(\xi_1)u + \mathcal{L}_{ff}(\xi_1)$. This yields the transverse feedback linearized system dynamics,

$$\dot{\xi} = \begin{bmatrix} \xi_2 \\ \mathcal{L}_{gf}(\xi_1)u + \mathcal{L}_{ff}(\xi_1) \end{bmatrix} \tag{2.3}$$

We would like to control the system using linear control, so we set $\xi_2 = \mathbf{kxi}$ and solve for the non-linear control signal u. We arrive at the control generation equation for transverse feedback linearization,

$$u = \frac{\mathbf{kxi} - \mathcal{L}_{ff}(\xi_1)}{\mathcal{L}_{gf}(\xi_1)} \tag{2.4}$$

The reader may wonder under what conditions $\mathcal{L}_{gf}(\xi_1)$ is non-zero. This is not apparent, although intuition may guide us. Since $\mathcal{L}_{gf}(\xi_1)$ is what scales the control signal, and therefore it may vanish whenever the control signal has no instantaneous impact on the change in the orthogonal error. This may occur, for example, when the differential robot is facing towards the nearest point.

2.2.2 Sylvester Approach

The Sylvester methodology relies on the theoretical developments of elimination theory. Elimination theory investigates the theoretical methods that discover common roots of polynomials. The essential idea is to reformulate Equation 2.1 into a system of homogenous linear equations that has a non-trivial kernel [10]. This non-trivial kernel allows for the derivation of an implicit representation of the path, otherwise known as a level set, that can then be stabilized by the transverse feedback linearization. First let us impose a regularity condition.

Assumption 2.2.1. *Path is self-intersection free.*

$$\forall \lambda_1, \lambda_2 \in \mathbb{R} : \sigma(\lambda_1) = \sigma(\lambda_2) \Leftrightarrow \lambda_1 = \lambda_2$$

Take notice that there is no restriction on λ in the assumption although work in this report does restrict the domain. Additionally, we need a result from linear algebra,

Theorem 2.2.2. *Given a system of n homogenous linear equations in n unknowns, $\mathbf{Ax} = 0$, non-trivial solutions x exist if and only if det $\mathbf{A} = 0$.*

Notice that if we formulate a new system from Equation 2.1 that is square and homogenous, we will be able to eliminate $\mathbf{x}$ by instead considering the det $\mathbf{A}$. This is otherwise known as finding the resultant. Homogenizing the path simply requires treating σ_1 and σ_2 as constants x_1 and x_2, moving them into the polynomial matrix $\mathbf{P}$. We call this modified matrix $\mathbf{Q}$. With this modification, we need only construct a square system out of $\mathbf{Q}$. In order to construct a square linear system, we consider multiplying both polynomials in the first and second component of σ by increasing powers of λ until we have generated a sufficient number of equations. For two quintics, we must multiply the polynomials by powers of λ up til λ^5, arriving at Equation 2.5.

$$\begin{bmatrix} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 6} \\ 0 & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 5} \\ \mathbf{0}_{1\times 2} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 3} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 4} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 2} \\ \mathbf{0}_{1\times 5} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 1} \\ \mathbf{0}_{1\times 6} & \mathbf{Q}_{1:} & \\ & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 6} \\ 0 & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 5} \\ \mathbf{0}_{1\times 2} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 3} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 4} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 2} \\ \mathbf{0}_{1\times 5} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 1} \\ \mathbf{0}_{1\times 6} & \mathbf{Q}_{2:} & \end{bmatrix} \begin{bmatrix} 1 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \\ \lambda^4 \\ \lambda^5 \\ \lambda^6 \\ \lambda^7 \\ \lambda^8 \\ \lambda^9 \\ \lambda^{10} \\ \lambda^{11} \end{bmatrix} = \mathbf{S} \begin{bmatrix} 1 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \\ \lambda^4 \\ \lambda^5 \\ \lambda^6 \\ \lambda^7 \\ \lambda^8 \\ \lambda^9 \\ \lambda^{10} \\ \lambda^{11} \end{bmatrix} \tag{2.5}$$

Carefully note that the tail of $\mathbf{Q}_{1:}$ and $\mathbf{Q}_{2:}$ contain a linear term in x_1 and x_2 respectively. Therefore if the matrix has a non-trivial kernel, i.e. its determinant vanishes as per Theorem 2.2.2, both x_1 and x_2 satisfy the polynomial equations. This motivates the definition of the zero level set, $s(x_1, x_2) = \det\ S$. This method assumes the matrix S doesn't have trivially vanishing determinant, such as if the leading coefficients of both polynomials σ_1 and σ_2 are equal. In that case, extensions to the method exist[10].

Now we can define ξ_1, the distance from the robot to the path, as the level set function $\xi_1 = (s \circ h)(\mathbf{x})$. Observe that ξ_1 is only dependent on x_1 and x_2. Define $\xi_2 = \dot{\xi}_1 = \nabla_{(x_1,x_2)} s\ \mathbf{Hf}(\mathbf{x})$. To complete the control system, we must differentiate ξ_2 in time. We expect Equation 2.3 to emerge naturally.

$$\begin{aligned} \dot{\xi}_2 &= \frac{d}{dt}\left[\nabla_{(x_1,x_2)} s\right] \mathbf{Hf}(\mathbf{x}) + \nabla_{(x_1,x_2)} s \frac{d}{dt}\left[\mathbf{Hf}(\mathbf{x})\right] \\ &= (\mathbf{Hf}(\mathbf{x}))^\intercal \frac{\partial}{\partial(x_1, x_2)}\left[\nabla_{(x_1,x_2)} s\right] \mathbf{Hf}(\mathbf{x}) + \nabla_{(x_1,x_2)} s \mathbf{H} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{g} u \\ &= \mathcal{L}_{ff} + \mathcal{L}_{gf} u \end{aligned}$$

The controller implementation follows from Section 2.2.1.

2.2.3 Serret-Frenet Frame Approach

First we must impose an extra regularity condition, in addition to Assumption 2.2.1 in order for this controller to work.

Assumption 2.2.3. *Existence and uniqueness of nearest point on the curve, the operating point.*

$$\begin{aligned} \forall \mathbf{h}(\mathbf{x}) : \exists \lambda_* \in [0,1], \epsilon > 0 : \\ &\mathbf{h}(\mathbf{x}) \in B_\epsilon\left(\sigma(\lambda_*)\right) \\ &\forall s \in \sigma(\lambda) \cap B_\epsilon\left(\sigma(\lambda_*)\right), s \neq \sigma(\lambda_*) : \|\mathbf{h}(\mathbf{x}) - \sigma(\lambda_*)\|_2 < \|\mathbf{h}(\mathbf{x}) - s\|_2 \end{aligned}$$

λ_* is defined as the operating point. It is not difficult to see that this point also has the property that $\sigma'(\lambda_*)$ is orthogonal to the error. A linear basis that spans $\mathbb{R}^2$ is built from λ_*. Let the first basis vector $\mathbf{e}_1 = \sigma'(\lambda_*)(\|\sigma'(\lambda_*)\|_2)$. The second basis vector can be formed through a 90 degree rotation, such that $\mathbf{e}_2 = \mathbf{R}_{\frac{\pi}{2}} \mathbf{e}_1$. Relying on the intuition of previous derivations, we consider $\xi_1 = \mathbf{e}_2{}^\intercal \left(\mathbf{h}(\mathbf{x}) - \sigma(\lambda_*)\right)$. In words ξ_1 is the orthogonal tracking error with the active operating point. We complete the dynamical system state by

differentiating ξ_1 to find ξ_2,

$$\begin{aligned}\xi_2 &= \dot{\xi}_1 \\ &= \dot{\mathbf{e}}_2^\intercal \left(\mathbf{h}(\mathbf{x}) - \sigma(\lambda_*)\right) + \mathbf{e}_2{}^\intercal \left(\mathbf{H}(\mathbf{f}(\mathbf{x}) + \mathbf{g}u) - \sigma'(\lambda_*)\dot{\lambda}\right) \\ &= \dot{\mathbf{e}}_2^\intercal \left(\mathbf{h}(\mathbf{x}) - \sigma(\lambda_*)\right) + \mathbf{e}_2{}^\intercal \left(\mathbf{H}(\mathbf{f}(\mathbf{x}) + \mathbf{g}u) - \sigma'(\lambda_*)\dot{\lambda}\right)\end{aligned}$$

We require a result from basic differential geometry in order to continue.

Lemma 2.2.4. *Let $\mathbf{v}_1(\lambda), \mathbf{v}_2(\lambda)$ be continuously differentiable functions that map to the unit circle in $\mathbb{R}^2$, such that $\mathbf{v}_2(\lambda) = \mathbf{R}_{\frac{\pi}{2}}\mathbf{v}_1(\lambda)$. Then, the derivatives of the vectors are related by a (parameter-varying) scalar multiple of the opposing vector such that*

$$\begin{aligned}\mathbf{v}_1'(\lambda) &= \kappa(\lambda)\mathbf{v}_2 \\ \mathbf{v}_2'(\lambda) &= -\kappa(\lambda)\mathbf{v}_1\end{aligned}$$

Proof. Observe that $\mathbf{v}_1{}^\intercal\mathbf{v}_1 = 1$ identically over the domain. Differentiating gives $\mathbf{v}_1'{}^\intercal\mathbf{v}_1 = 0$. Since $\{\mathbf{v}_1, \mathbf{v}_2\}$ spans $\mathbb{R}^2$, $\mathbf{v}_1'{}^\intercal = \kappa(\lambda)\mathbf{v}_2$. This choice isn't unique but is convenient. It is worth mentioning that κ must be continuous at minimum. Further restrictions would be imposed by restrictions on the original functions. The derivative $\mathbf{v}_2'$ follows naturally from substituting $\mathbf{v}_2(\lambda) = \mathbf{R}_{\frac{\pi}{2}}\mathbf{v}_1(\lambda)$. □

A generalization of Lemma 2.2.4 exists in higher dimensions and can be found in any modern differential geometry text[5]. This lemma permits the elimination of the first and last terms in ξ_2. In practice, it may not always be possible to use — or even find — the closest point λ_* and therefore there may be some error introduced in the system dynamics. An analysis of this effect is performed later. For now, we discard this term and observe that the control signal is in the kernel of $\mathbf{H}$ yielding a simple definition for ξ_2,

$$\xi_2 = \mathbf{e}_2{}^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})$$

This definition may be intuitively described as the projection of the observable system dynamics on the orthogonal vector. Define a new state vector $\xi = (\xi_1, \xi_2)^\intercal$. Once again we would like to take the time derivative of ξ_2 to complete the dynamical system. However two more Lemmas are required to help us take the time derivative of the basis vectors.

Lemma 2.2.5. *There exists a function $\varpi : \mathbf{h} \mapsto \lambda$ that generates the operating point λ_* from the position of the robot (the observation). The derivative of ϖ with respect to the observation is*

$$\dot{\varpi}(\mathbf{h}) = \frac{\mathbf{e}_1{}^\intercal}{\|\sigma'(\lambda_*)\|_2}$$

Proof. Refer to Gill [3]. □

Lemma 2.2.6. *The curvature, $\kappa(\lambda)$, is*

$$\kappa(\lambda) = \frac{\sigma''(\lambda)^\intercal\mathbf{e}_2}{\|\sigma'(\lambda)\|_2}$$

Proof. Differentiate $\mathbf{e}_1$ with respect to λ explicitly,

$$\begin{aligned}
\frac{d\mathbf{e}_1}{d\lambda} &= \frac{d}{d\lambda}\left[\frac{\sigma'(\lambda)}{\|\sigma'(\lambda)\|_2}\right] \\
&= \frac{1}{\|\sigma'(\lambda)\|_2^2}\left[\sigma''(\lambda)\left\|\sigma'(\lambda)\right\|_2 - \sigma'(\lambda)(\left\|\sigma'(\lambda)\right\|_2)^{-1}\sigma''(\lambda)^\intercal\sigma'(\lambda)\right] \\
&= \frac{1}{\|\sigma'(\lambda)\|_2}\left[\sigma''(\lambda) - (\sigma''(\lambda)^\intercal\mathbf{e}_1)\mathbf{e}_1\right] \\
&= \frac{1}{\|\sigma'(\lambda)\|_2}\left[\sigma''(\lambda)^\intercal\mathbf{e}_2\right]\mathbf{e}_2
\end{aligned}$$

Likening the result with Lemma 2.2.4, we see the statement is proven. □

Computing the derivative of ξ_2 we find,

$$\begin{aligned}
\dot{\xi}_2 &= \dot{\mathbf{e}}_2^\intercal H\mathbf{f}(\mathbf{x}) + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\left[\mathbf{f}(\mathbf{x}) + \mathbf{g}u\right] \\
&= (H\mathbf{f}(\mathbf{x}))^\intercal\dot{\mathbf{e}}_2 + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\left[\mathbf{f}(\mathbf{x}) + \mathbf{g}u\right] \\
&= -(H\mathbf{f}(\mathbf{x}))^\intercal\mathbf{e}_1\kappa(\lambda_*)\frac{\mathbf{e}_1{}^\intercal H\mathbf{f}(\mathbf{x})}{\|\sigma'(\lambda_*)\|_2} + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{g}u \\
&= -(H\mathbf{f}(\mathbf{x}))^\intercal\mathbf{e}_1\frac{\sigma''(\lambda)^\intercal\mathbf{e}_2}{\|\sigma'(\lambda)\|_2}\frac{\mathbf{e}_1{}^\intercal H\mathbf{f}(\mathbf{x})}{\|\sigma'(\lambda_*)\|_2} + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{g}u \\
&= -\left[\mathbf{e}_1{}^\intercal H\mathbf{f}(\mathbf{x})\right]^2\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{g}u \\
&= \mathcal{L}_{ff}\xi_1 + \mathcal{L}_{gf}\xi_1 u
\end{aligned}$$

This result completes the dynamical system and allows for the construction of the linear controller. However, we would like to simplify the above system to be in terms of only ξ, if possible. A useful result that combines the knowledge of our model and formulation of the basis vectors is required.

Lemma 2.2.7. *The velocity of the robot may be decomposed through projections on the basis vectors,*

$$v = \sqrt{\left(\mathbf{e}_2{}^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})\right)^2 + \left(\mathbf{e}_1{}^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})\right)^2}$$

Which implies that,

$$\left(\mathbf{e}_1{}^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})\right) = \sqrt{v^2 - \xi_2^2}$$

$$
\begin{aligned}
\dot{\xi}_2 &= -\left[\mathbf{e}_1{}^\intercal H\mathbf{f}(\mathbf{x})\right]^2 \frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{g}u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal H\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{g}u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal\begin{bmatrix}-v\ \sin x_3\\ v\ \cos x_3\end{bmatrix}u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal\mathbf{R}_{\frac{\pi}{2}}\begin{bmatrix}v\ \cos x_3\\ v\ \sin x_3\end{bmatrix}u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_2{}^\intercal\mathbf{R}_{\frac{\pi}{2}}\mathbf{H}\mathbf{f}(\mathbf{x})u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \left(\mathbf{R}_{-\frac{\pi}{2}}\mathbf{e}_2\right)^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + \mathbf{e}_1{}^\intercal\mathbf{H}\mathbf{f}(\mathbf{x})u \\
&= -\left(v^2-\xi_2^2\right)\frac{\mathbf{e}_2{}^\intercal\sigma''(\lambda)}{\sigma'(\lambda)^\intercal\sigma'(\lambda_*)} + u\sqrt{v^2-\xi_2^2}
\end{aligned}
$$

Further simplification is not necessary as it is clear that $|\xi_2|$ must never equal v. Intuitively this means that the robot's speed must not be completely represented in the orthogonal direction – towards the path. This concern motivates an analysis on whether stability may be guaranteed within a bounded volume that does not intersect the $|\xi_2| = v$ isoclines.

Stability Analysis and Control Design

The final control system, with proportional linear feedback, is

$$\dot{\xi} = \begin{bmatrix}0 & 1\\ g_1 & g_2\end{bmatrix}\xi \tag{2.6}$$

We begin with a necessary result from Dynamical System Theory.

Theorem 2.2.8 ((General) Lyapunov's Method)**.** *Let ξ_* be an equilibrium point of Equation 2.6, $\Omega \subset \mathbb{R}^2$ be an open set containing ξ_*, and $V : \Omega \to \mathbb{R}$ be continuously differentiable such that*

1. *$V(\xi_*) = 0, V(\xi) > 0 \Leftrightarrow \xi \neq \xi_*$*
2. *$\dot{V}(\xi) \leq 0, \xi \in \Omega$*
3. *The set $E = \left\{\xi \in \Omega : \dot{(}V(\xi)) = 0\right\}$ does not contain any whole orbits except ξ_* (LaSalle's Invariance Principle)*

Then all orbits in Ω asymptotically converge to ξ_*.

Proof. See Liu [7]. □

Now we prove the main result,

Theorem 2.2.9 (Arbitrarily Bounded Stability of Serret Frenet Control)**.** *For any $M \in \mathbb{R}$ and $M > 0$, an Ω may be constructed such that,*

1. $\sup_{\xi\in\Omega}|\xi_1| = M$
2. $\sup_{\xi\in\Omega}|\xi_2| = v$

Let $g_2 < 0$. Then, there exists a Lyapunov function $V : \Omega \to \mathbb{R}$ and gain g_1 that satisfies the requirements of Theorem 2.2.8 such that the dynamical system is guaranteed to converge to the path.

Proof. We use the method of undetermined coefficients over a quartic positive definite form. Let $V(\xi) = \xi_1^4 + a\xi_1^2\xi_2^2 + b\xi_2^4$. Choose $\Omega = \{\xi : V(\xi) = D\}$ for some $D \in \mathbb{R}$. Choose $b = Dv^{-4}$. Then,

$$\begin{aligned}
\dot{V} &= 4\xi_1^3\dot{\xi}_1 + 2a\left[\xi_1\dot{\xi}_1\xi_2^2 + \xi_1^2\xi_2\dot{\xi}_2\right] + 4b\xi_2^3\dot{\xi}_2 \\
&= 4\xi_1^3\xi_2 + 2a\left[\xi_1\xi_2^3 + g_1\xi_1^3\xi_2 + g_2\xi_1^2\xi_2^2\right] + 4bg_1\xi_1\xi_2^3 + 4bg_2\xi_2^4 \\
&= \left(4 + 2ag_1\right)\xi_1^3\xi_2 + 2\left(a + 2bg_1\right)\xi_1\xi_2^3 + \left(2a\xi_1^2\xi_2^2 + 4b\xi_2^4\right)g_2 \\
&= \left(4 + 2ag_1\right)\xi_1^3\xi_2 + 2\left(a + 2Dv^{-4}g_1\right)\xi_1\xi_2^3 + \left(2a\xi_1^2\xi_2^2 + 4Dvd^{-4}\xi_2^4\right)g_2
\end{aligned}$$

Impose the requirement that cross-terms vanish identically. This gives restrictions on a and g_1.

$$\begin{aligned}
g_1 &= -\frac{v^2}{\sqrt{D}} \\
a &= 2\frac{\sqrt{D}}{v^2}
\end{aligned}$$

This pair uniquely defines V as $V(\xi) = \xi_1^4 + 2\frac{\sqrt{D}}{v^2}\xi_1^2\xi_2^2 + \frac{D}{v^4}\xi_2^4$. Interestingly, this is actually the square of a quadratic form. Specifically,

$$V(\xi) = \left[\xi_1^2 + \left(\frac{\sqrt{D}}{v^2}\right)\xi_2^2\right]^2$$

We can now see that the conditions on Ω are in fact observed with $D = M^4$. However, we still need to prove asymptotic stability. We now know that $\dot{V} = \left(2a\xi_1^2\xi_2^2 + 4Dv^{-4}\xi_2^4\right)g_2$. This function is zero on the $\xi_2 = 0$ axis. However, we note that since $g_1 \neq 0$, there are no equilibria on this axis. More importantly, no other whole orbits exist on this axis except for the equilibrium point, $\xi = \mathbf{0}$. By Theorem 2.2.8, the system is asymptotically stable as required. □

This result motivates a control design methodology for the Serret-Frenet controller. By choosing a maximum allowable orthogonal error M, the gain $g_1 = -(\frac{v}{M})^2$ is determined. Then overshoot requirements may be satisfied by choice in g_2. We choose to have critical damping, mandating $g_2 = -2\frac{v}{M}$. Observe that as $M \to 0$, that the gains are unbounded. This is as one would expect.

Chapter 3

Planning

Path planning involves the generation of waypoints and curves for the purpose of arriving at some goal region, $\mathcal{G}$. This paper only describes the Rapidly exploring Randomized Tree (RRT) algorithm. A majority of the investigation in planning occurred in the polynomial spline generation phase.

3.1 RRT

The Rapidly exploring Randomized Tree (RRT) algorithm is made up of the following steps [4],

1. Initialize tree $R = \{x_0\}$ with the initial point.
2. Randomly sample a candidate point $x \in \mathcal{T}$.
3. Find the nearest point $x_n \in R$ to x.
4. Add an edge (x_n, x) if the path between them is collision free.
5. If $x \in \mathcal{G}$ halt, otherwise repeat from (2).

The tree grows in a random fashion from the initial point, rapidly expanding in $\mathcal{T}$ until the goal can be reached. Once the algorithm completes, the programmer need only back-trace from the goal point towards the initial point; trees have a unique path from any node to the root. There are a few design decisions that can be made here in order to improve performance of the algorithm. One is to store obstacles in what is known as an R-Tree. R-Trees — a tree whose name stems from rectangle, not rapidly nor randomized — are a geometrical data structure that stores data according to their position in space. R-Tree creates bins that store objects with bounding rectangles that are near one another. This data structure has an *average* time complexity that is $\mathcal{O}(\log_2 N)$ for searching an element in the two dimensional case. As a result, it scales well even for large spatial domains where many obstacles may exist.

One reason this algorithm was developed was to help developers effectively incorporate constraints on the generated path. Recall that the effectiveness of the transverse feedback linearization is related to the curvature of the path. The higher the curvature, the more sensitive the controller is to small errors. Moreover, it may be the case that Assumption 2.2.3 fails to hold. One way to easily restrict the curvature is to ensure the angle of edge (x_n, x) doesn't deviate strongly from the angle of the line (x_m, x_n). That is, the angle between any two successive edges is restricted to $\pi \pm \theta_{lim}$.

Another approach attempted is to enforce a distance constraint that is related to the angle. Unfortunately, this does not work as well in practice, as it greatly reduces the exploration performance of RRT.

RRT also does not by itself provide guarantees about the 'directness' of the path generated. This can changed by creating a bias towards sampling near $\mathcal{G}$[6][1]. We chose to randomly sample candidate points from a truncated-normal distribution centred about $\mathcal{G}$. The variance was chosen to ensure about 70-95% of points are within the task space. This allowed for biased sampling that still allowed for exploration of the domain.

3.2 Polynomial Splining

Polynomial splining is a technique by which possibly distinct polynomials are stitched together to preserve a certain level of continuity at the stitch-points. Consider the polynomial $\sigma_i = \sum_{k=0}^{N} (a_{ik}, b_{ik})^\intercal \lambda^k$. Normally we would like this polynomial to evaluate to the start and end points at $\lambda = 0$ and $\lambda = 1$ respectively. This clearly enforces a linear equality constraint. Continuity conditions are imposed on other polynomials that touch this polynomial, at the end points, by first enforcing that they evaluate to the same point at some other λ (possibly the same value). Then derivatives are taken, as many as the order of continuity is required, and evaluated at the endpoints and set equal to one other. This results in a homogenous linear equation. This set of linear equations may be solved by any standard algorithm, such as LU decomposition.

In the case where there are fewer equations than unknowns, the system is under-determined. Although this can be solved using Gaussian elimination, it is preferred to use the Moore-Penrose pseudo-inverse (least squares) instead. This finds a solution that also minimizes the norm of the vector, in this case the vector of coefficients. This is important, as it means the curve shouldn't exhibit larger curvature than necessary.

There is a problem with the above approach in the context of path planning. Since these polynomials are generated from waypoints generated by RRT, we know that the only guarantees we have about the waypoints are that the edges between them are collision free. There is no reason to think that the polynomial may not deviate from this edge, which it normally does, in a manner that may collide with an object. Of course, the easy way to deal with this is to test manually whether the polynomial intersects the object through a linear approximation of it. This is slow.

Instead, we design a new methodology that attempts to fit the polynomial to the edges, instead of restricting the polynomial to the waypoints.

3.2.1 Loosely Constrained Spline

Let W be the set of waypoints and let the polynomial be defined as before. We would like to construct a polynomial that fits every point W within an error δ while minimizing the deviation between it and the line between the waypoints. A simple way of performing this is to parameterize the edges with the same parameter as the polynomial, and construct a least squares matrix for these intermediate points. Conditions may be constructed in the form of $\sigma(\lambda) - w_i < \delta$ and $w_i - \sigma(\lambda) < \delta$. This form allows for the construction of a linear-inequality constrained least squares problem. This can be solved in MATLAB using `lsqlin`; any optimization algorithm that allows for constraints may perform well in this area.

By allowing for this much freedom, the curve closely resembles the line and is unlikely to introduce collisions in the generated path. Of course, the path does deviate from the waypoint and this is the cost of this method. Further, the cost of the splining rises dramatically as a linear algebra problem is now an optimization problem.

Chapter 4

Case Study

4.1 Baseline Study

An initial performance comparision on the controllers was performed in MATLAB with a quintic polynomial path generated using the technique described in Section 3.2.1. The polynomial has small coefficients with enough curvature to have interesting behaviour but not too high curvature that may adversely affect all controllers. The path can be seen in both Figure 4.2 and Figure 4.1. The tracking controller, as seen in Figure 4.1, seems to successfully track the path.

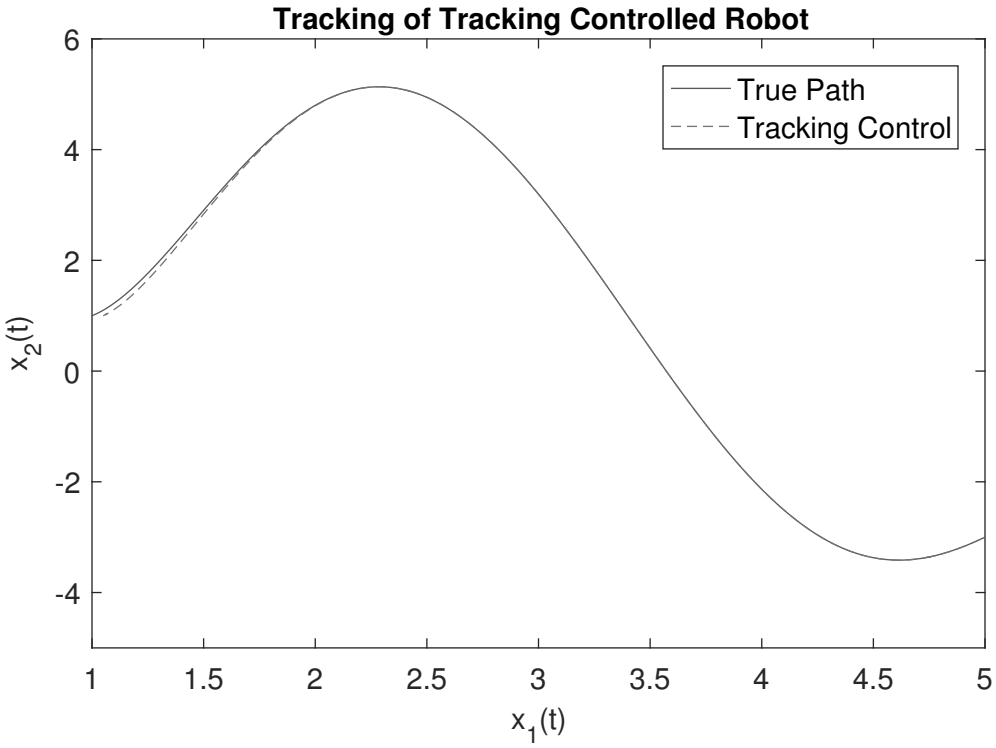


Figure 4.1: The track of the robot controlled by Tracking Control in task space. Driving towards and onto path.

The Serret-Frenet and Sylvester controllers should exhibit exponential convergence to the path; the gains are chosen so as to guarantee exponential stability in ξ. Gains are chosen to ensure the Serret-Frenet controller doesn't exceed an orthogonal tracking error of 0.1. The Sylvester approach uses the exact same gains to provide a fair comparison. Figure 4.2 shows the resulting path of both robots in a situation where they are on the path but not facing in the right direction — the robots start with a heading error of over 45 degrees. Notice that

both methodologies converge and remain on the path no matter the behaviour of the curve.

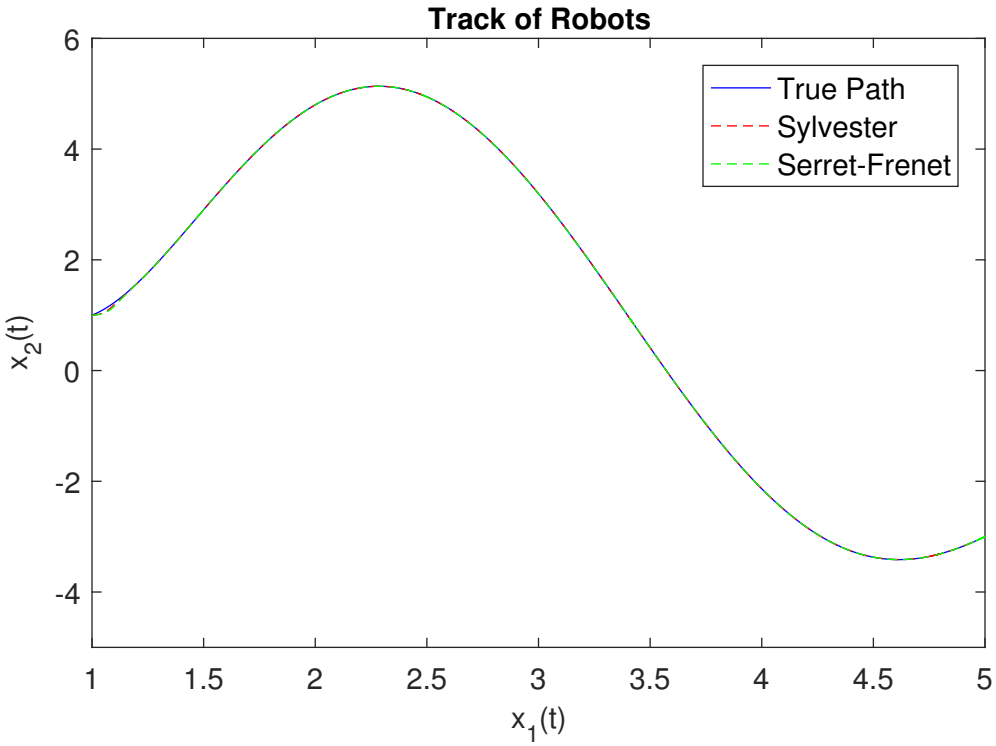


Figure 4.2: The track of the robots in task space driving towards and on the path.

Figure 4.2 isn't precise enough to verify performance, so we turn to the orthogonal tracking error plot in Figure 4.3. It is now apparent how far away both methods go before converging to the path tangentially. Interestingly, the Sylvester controller outperforms the Serret-Frenet controller in both maximum deviation and rate of convergence. At face value, this indicates that the Sylvester controller is the one to choose.

Figure 4.4 evaporates any such deduction. The Sylvester controller generates states that are orders of magnitude larger than the Serret-Frenet controller, which in turn means the Sylvester controller generates extremely large control signals. Keep in mind that the error in position was less than 10^{-1}! This is impractical. Furthermore, this baseline test was only done for a simple polynomial with small coefficients. As a result, there is no reason to expect this method will extend well for large polynomial over a large domain.

This isn't the only dimension where the Sylvester controller falls short. Table 4.1 collates the time-cost of executing the control-generation functions for both controllers, measured in MATLAB. The gradient descent algorithm is included so as to provide a useful baseline of performance. Notice that the gradient descent is the dominating cost in the Serret-Frenet controller. The Sylvester controller, on the other hand, performs poorly. In fact, data — not shown here — indicates that 62% of the time is spent just calculating the ξ state vector. Any implementation of the Sylvester controller can perform, at best, around 30 steps per second. The reader would like to take note that the machine used for benchmarking uses an Intel processor running at 2.50GHz. An embedded processor is unlikely to effectively execute the Sylvester controller. Unfortunately, these are timing benchmarks and are, as a result, not easily transferable to other hardware. An analysis of floating point operations is required.

Both controllers were implemented in C++ and compiled with GCC with optimizations turned off in order to correctly estimate number of floating point operations. Table 4.2 lists the results. The Serret-Frenet controller is expected to be 3 orders of magnitude faster just by a rough count of floating point operations. This is a rough estimate as it doesn't account for optimizations.

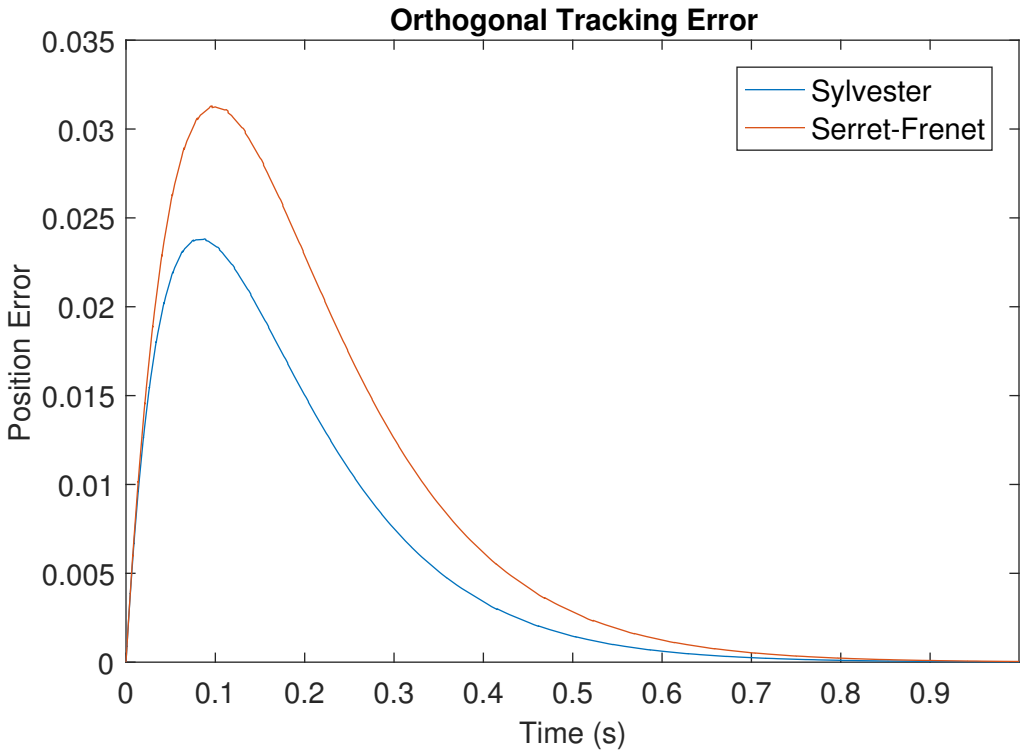


Figure 4.3: Comparison of tracking error. Notice that both experience exponential convergence, however the Sylvester method seems to converge faster.

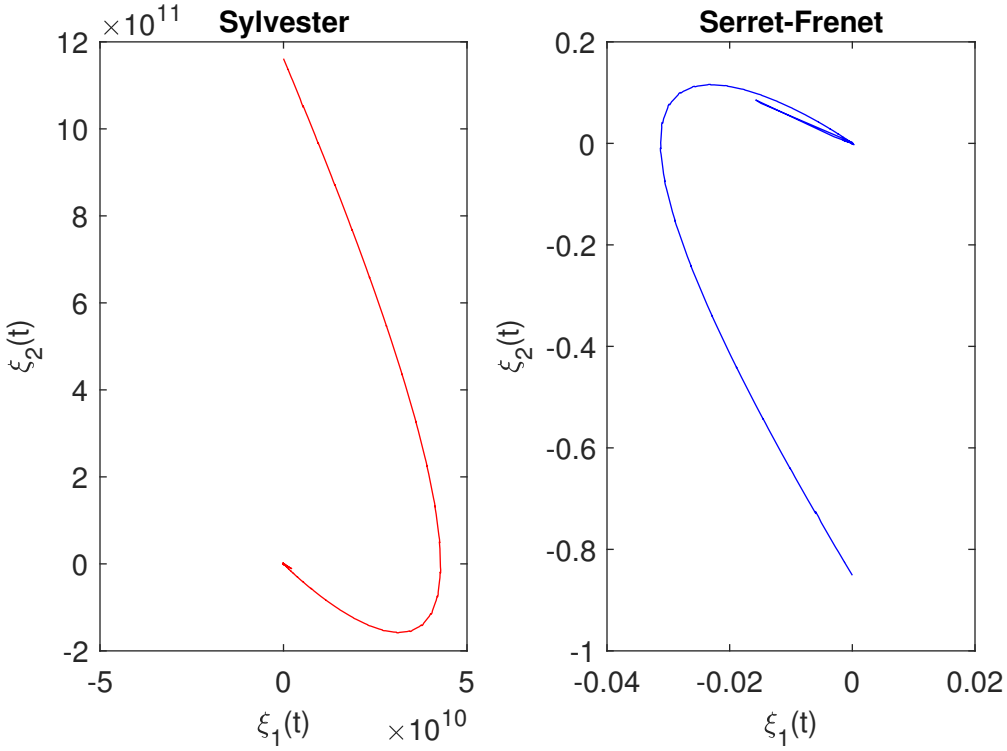


Figure 4.4: The linearized state ξ for both controllers. Take careful note of the scale of the axes!

Function of Interest	Total Time Used (s)	Number of Calls	Estimated Cost (ms)
Gradient Descent	11.339	2017	5.6217
Serret-Frenet	12.812	2017	6.3520
Sylvester	61.338	2095	29.278

Table 4.1: Time cost of the control generation functions. The gradient descent algorithm cost is included in their timing results.

Function of Interest	Floating Point Operation Count	Comments
Serret-Frenet	316	From evaluating polynomial and its derivatives.
Sylvester	111159	From the level set evaluation functions.

Table 4.2: Floating point operation counts. Manually counted by assembly inspection and regular expression searches, cross-verified with actual source code. Values may vary by single-digit percentiles but their order of magnitude should remain the same.

4.2 Mock World Study

In this section, the controllers are pitted against one another in a mock real world use case. The University of Waterloo campus map was mocked, treating buildings as rectangular obstacles, as shown in Figure 4.5. The robots begin at the bottom of campus and are told to make their way to the top of campus. An implementation of RRT is used to discover paths to the goal. The spline generated is a sequence of quintic polynomials stitched together preserving C^2 continuity as required. The path followed by both the Serret-Frenet controller and the Tracking controller are shown in Figures 4.6 and 4.7. The asterisks occur at points where the robot made the decision to change the path, i.e. initiate path planning.

Both controllers achieve, at face value, the tracking objective. We can only know for sure by verifying the tracking error. Figures 4.8 and 4.9 illustrate the tracking error. The Serret-Frenet controller maintains an almost constant error that is negligible, since the path is planned such that the robot is already correctly configured on the path. This corroborates our knowledge that the Serret-Frenet controller treats the path as invariant. The tracking controller also achieves relatively good tracking, except at the path planning event points. This may occur because the path plan begins at the robot's position, instead of the lead point. Barring this issue, the tracking controller performs well. However, notice that we can achieve just as good tracking without the velocity control introduced by the tracking controller. Moreover, the tracking controller requires a great deal of tuning.

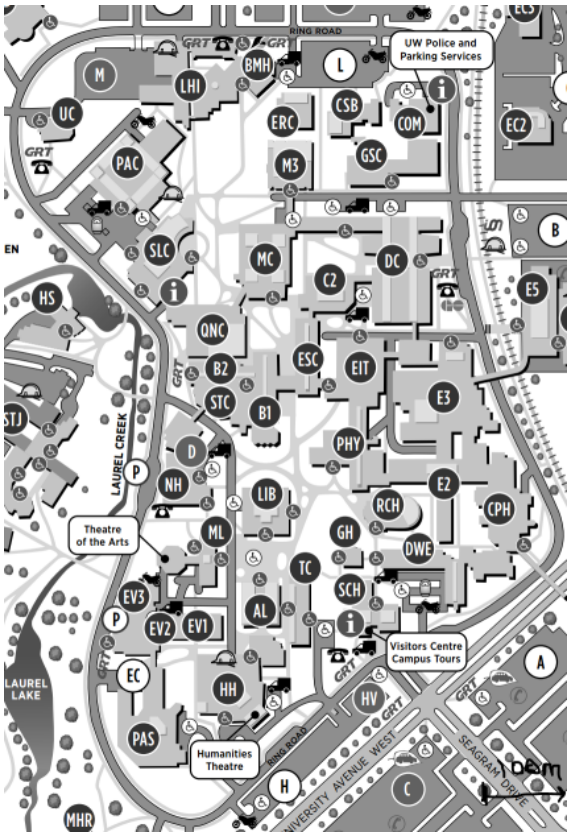
The reader may be wondering why the Sylvester controller was not tested in this benchmark. There are a few reasons for this:

1. The control level set functions could not be optimized by the compiler (GCC).
2. The control signals generated were excessive.
3. Integration tolerances may not have been achievable.

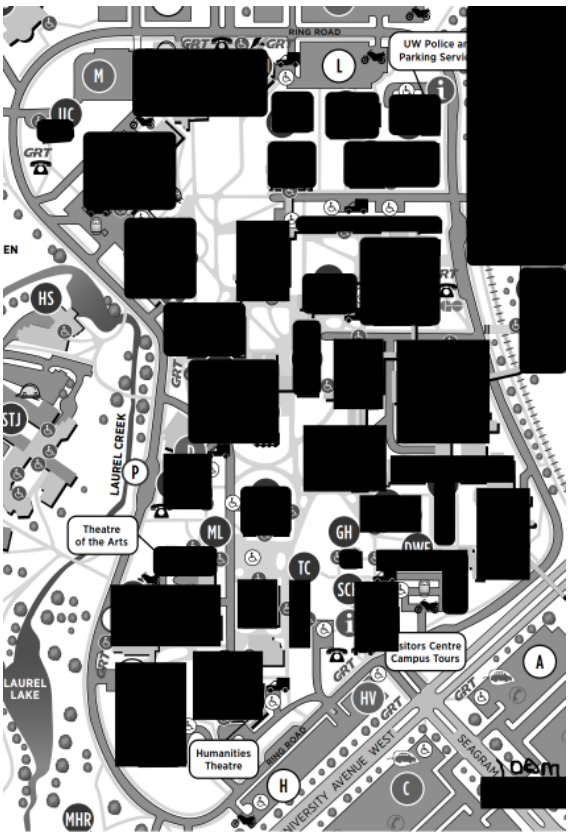
The orthogonal tracking error for the Sylvester controller, before the simulation stalled, is shown in Figure 4.10. The values are 37 orders of magnitude larger than in the Serret-Frenet controller (or, for that matter, any reasonable controller). The control signal generated is proportional to a value of the same order of magnitude, and so this controller is effectively rendered impractical.

4.3 Conclusions

A set of path following controllers were implemented in both MATLAB and C++ and verified in a relatively large scale mock example. A reformulation of the splining problem in terms of looser constraints was proposed.



(a) Original campus map.



(b) Original campus map with rectangular obstacles.

Figure 4.5: Mock campus world model. 1 pixel equals approximately 0.64m.

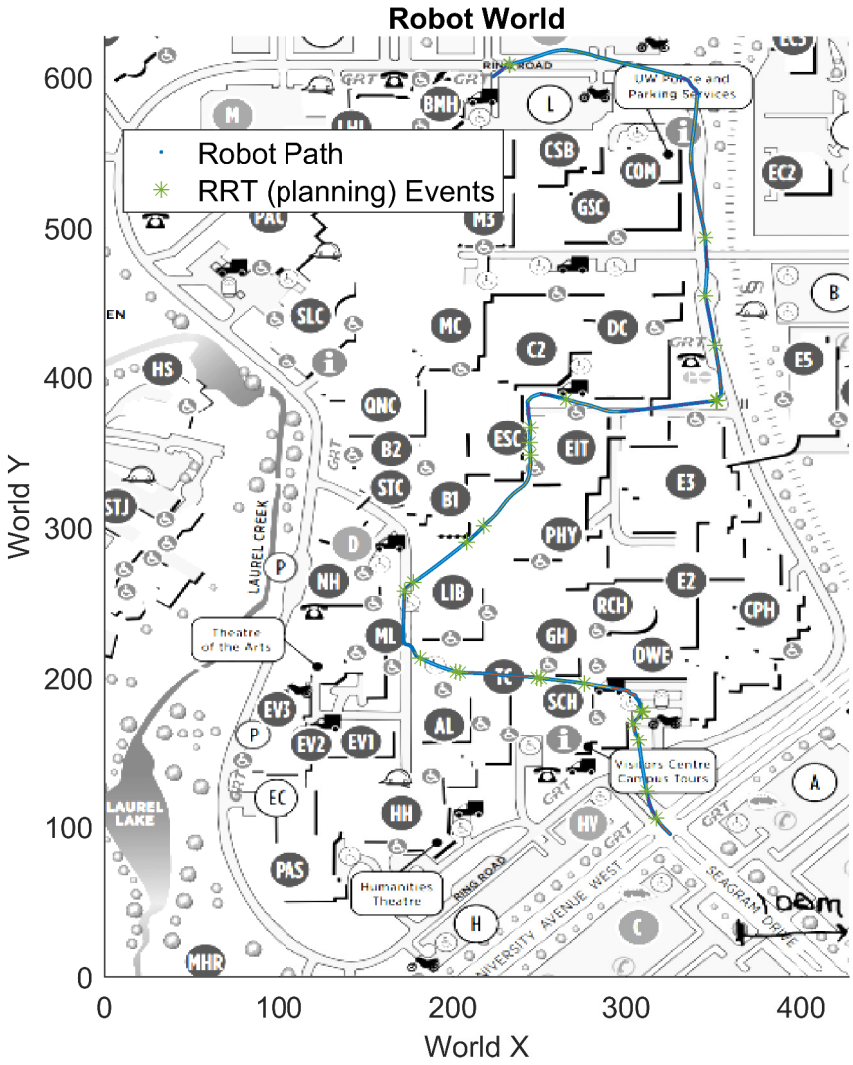


Figure 4.6: Path followed by Serret-Frenet controller.

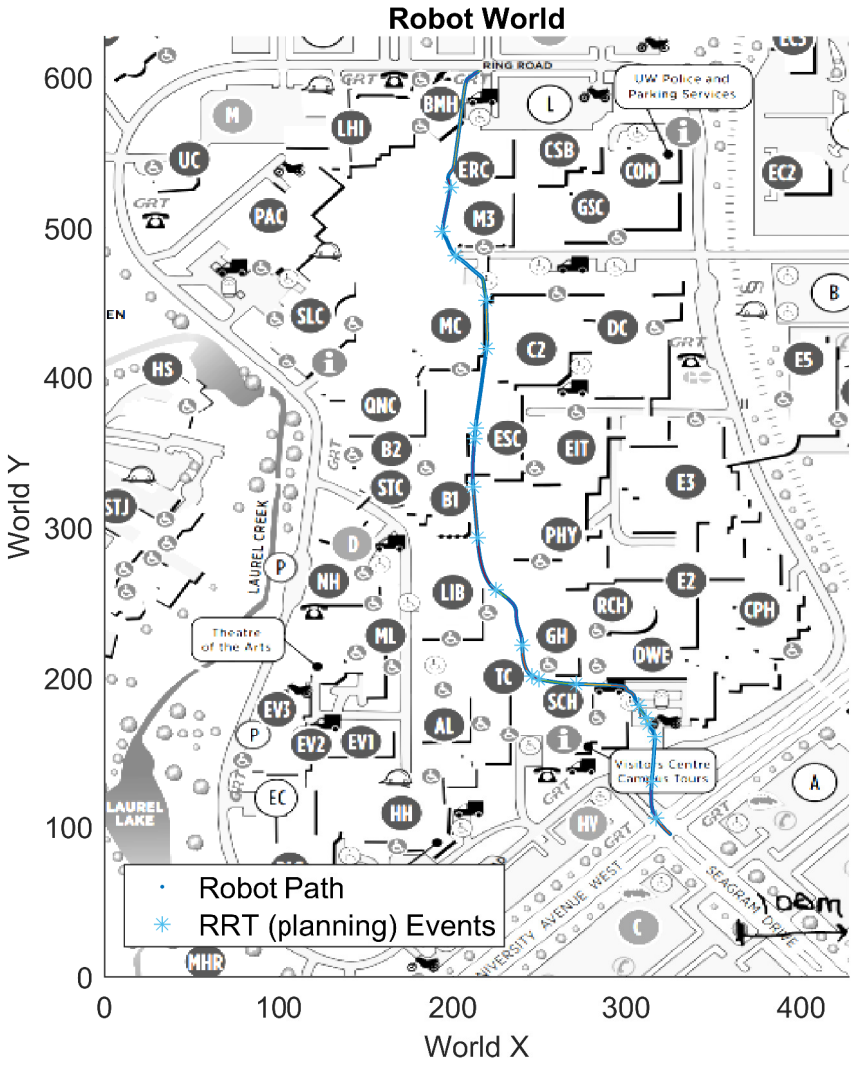


Figure 4.7: Path followed by Tracking controller.

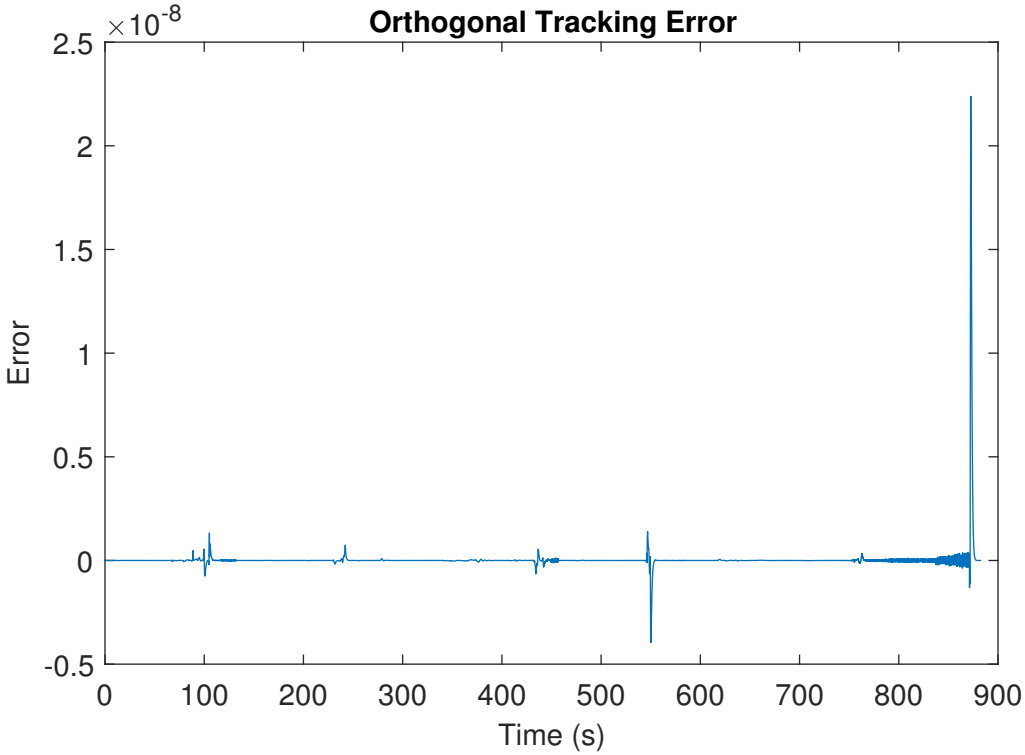


Figure 4.8: Error (orthogonal only) in Serret-Frenet controller.

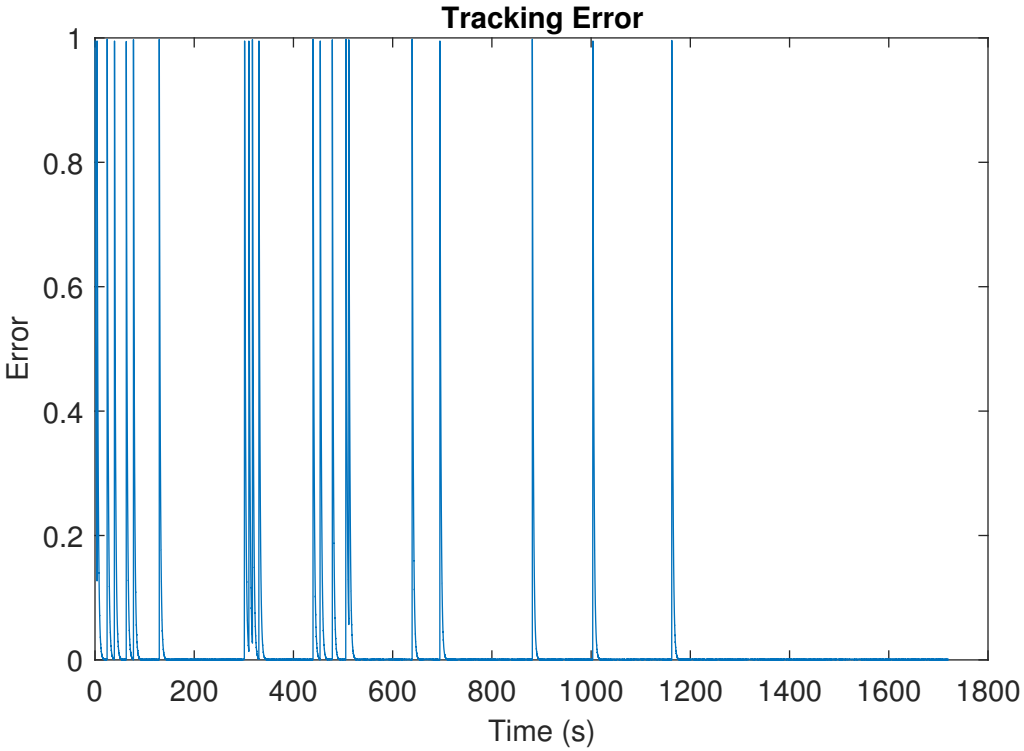


Figure 4.9: Error in Tracking controller.

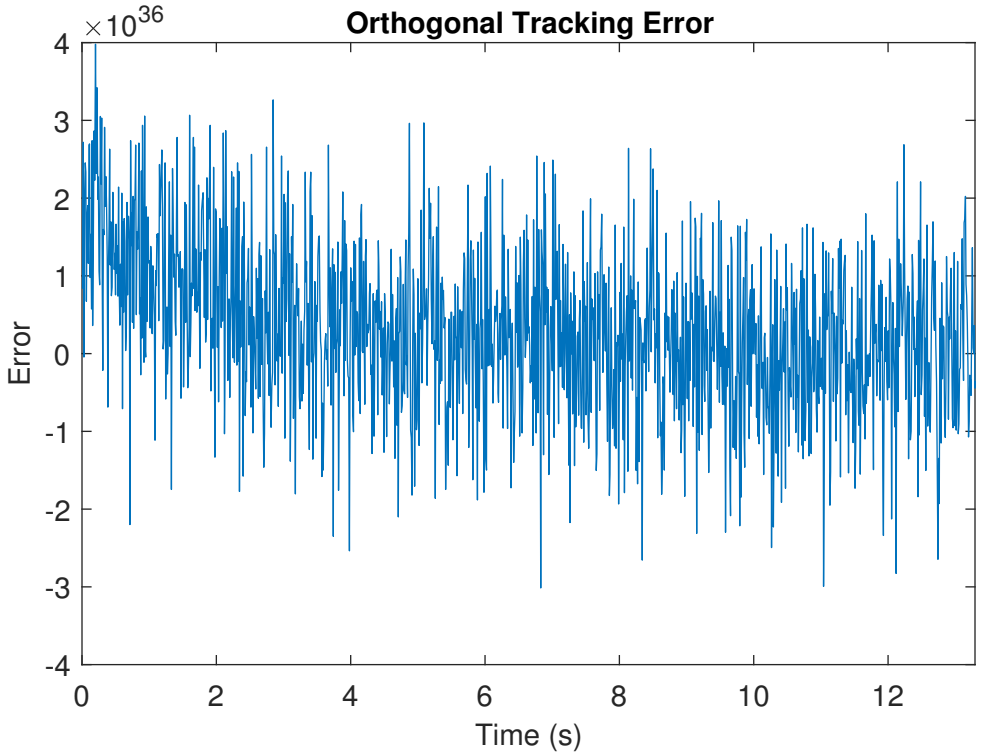


Figure 4.10: Error in Sylvester controller. Note the wild oscillations and relatively high magnitude error.

An analysis of the Serret-Frenet controller was performed, with the additional development a control design methodology that allows for a control engineer to design gains based on a-priori restrictions on the orthogonal tracking error.

Theorem 2.2.9 does provoke a few interesting lines of research. Firstly, it motivates the use of a dynamic gain, that changes as the system evolves deeper into the invariant set. This approach may be able to speed convergence towards the path while ensuring the robot doesn't veer outside into the unknown domain. In addition to this, the Theorem seems to indicate that a design methodology may exist that incorporates conditions on ξ_2 as well! Restrictions on ξ_2 are effectively equivalent to preventing the robot from approaching too steeply towards the path. This may have applicability in domains where the robot cannot (and maybe even should not) make a bee-line towards the path for safety reasons. A simple example is a car which must switch from the slow lane to the fast lane on a wide highway. Even on an open road, it is safer for both passengers and other drivers to approach in a manner that reduces the turn rate.

The Sylvester controller, though mathematically elegant, doesn't seem to be well suited for practical implementaion. In fact, my compiler couldn't optimize any of the Lie derivative and Level set functions for the Sylvester controller. Instead, the compiler hung. This and the fact that the Sylvester controller is expensive in terms of floating point operations, indicates a lack of practicality. Even in simulation, the Sylvester controller generated unreasonable control signals and linearized states. It is apparent that the Serret-Frenet controller is superior.

References

[1] K. Yang et al. "Spline-Based RRT Path Planner for Non-Holonomic Robots". In: *J Intell Robot Syst* 73 (2014), pp. 763–782.

[2] *Boost C++ Libraries*. http://www.boost.org/.

[3] R.J. Gill, D. Kulic, and C. Nielsen. "Spline Path Following for Redundant Mechanical Systems". In: *IEEE Transactions on Robotics* 31 (2015), pp. 1378–1392.

[4] S. Karaman and E. Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: *The International Journal of Robotics Research* 30 (7 2011), pp. 846–894.

[5] W. Kuhnel. *Differential Geometry: Curves—Surfaces—Manifolds Second Edition*. American Mathematical Society, 2006.

[6] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[7] X. Liu. *Introduction to Dynamical Systems — Course Notes for AMATH 451*. University of Waterloo, 2012.

[8] Phil Nash. *Catch Test Framework*. https://github.com/philsquared/Catch.

[9] C. Nielsen and M. Maggiore. "Output stabilization and maneuver regulation: A geometric approach". In: *System Control Letters* 55 (2006), pp. 418–427.

[10] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. "Implicit Representation of Parametric Curves and Surfaces". In: *Computer Vision, Graphics and Image Processing* 28 (1984), pp. 72–84.